(2)

Task/Subtask ID52.1(2)
CDRL Sequence 05504-001
31 July 1993

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Cleanroom Engineering Handbook
Volume 6
Certification Team Practices

Contract No. F19628-88-D-0032

Task ID52 – STARS Technology Transfer Demonstration Project for the U.S. Army

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

Task/Subtask ID52.1(2)
CDRL Sequence 05504-001
31 July 1993

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Cleanroom Engineering Handbook
Volume 6
Certification Team Practices

Contract No. F19628-88-D-0032

Task ID52 – STARS Technology Transfer Demonstration Project for the U.S. Army

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	7/31/93	Initial

4. TITLE AND SUBTITLE
Cleanroom Engineering Handbook:
Certification Team Practices

5. FUNDING NUMBERS
F19628-88-C-0032/0010

6. AUTHOR(S)
Ara Kouchakdjian
Richard H. Cobb
Alan R. Hevner
James A. Whittaker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

SET, Inc.
2770 Indian River Blvd.
Vero Beach, FL 32960

8. PERFORMING ORGANIZATION REPORT NUMBER
05504-001
Volume 6

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Center/ENS
Air Force Materiel Command, USAF
5 Eglin Street, Building 1704
Hanscom Air Force Base, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Cleared for Public Release, Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This is one of a series of six engineering handbooks prepared for and used by the engineering staff at Picatinny Arsenal for the STARS technology transfer demonstration. The handbooks define the engineering process and algorithms that will be used in Cleanroom projects. They are designed to provide support to trained engineers using Cleanroom Engineering, not to substitute for training.

The Cleanroom process model for software system development projects is presented in Volume 1 - Cleanroom Process Overview - of this series of Cleanroom Process Manuals. This handbook, Volume 6, describes the activities of Cleanroom softfware certification for each increment/accumulation of project development. Process P5.j.2, Preparation for Certification of Accumulation j and process P6.j, Software Certification for Accumulation j, are expanded into engineering subprocesses (i.e., tasks) for execution by Cleanroom-trained software engineers. These tasks result in the preparation for and performance of the certification of developed software.

The reader of this volume should be familiar with the information and terms described in Volume 4 - Specification Team Practices. This handbook, Volume 6, builds on the theory involving software usage modeling which is introduced in Volume 4.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Certification, Cleanroom, Cleanroom Engineering, Development, Management, Software Development, Specification	54
	16. PRICE CODE N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

PREFACE

This series of handbooks is prepared for use by managers and engineers assigned to Cleanroom projects at Picatinny Life Cycle Software Engineering Center.

These handbooks define the engineering process and algorithms that will be used in Cleanroom projects.

This document was developed by the IBM Federal Systems Company, located at 800 North Frederick Avenue, Gaithersburg, MD 20879 and Software Engineering Technology, Inc. located at 2770 Indian River Boulevard, Vero Beach, FL 32960. Questions or comments should be directed to Mr. Paul Arnold at 301-240-7464 (Internet: pga@sei.cmu.edu).

CERTIFICATION TEAM PRACTICES

TABLE OF CONTENTS

CERTIFICATION TEAM PRACTICES

SECTION 1: INTRODUCTION

The Cleanroom process model for software system development projects is presented in *Volume 1 - Cleanroom Process Overview* of this series of Cleanroom Process Manuals. This manual, Volume 6, describes the activities of Cleanroom software certification for each increment/accumulation of project development. Process P5.j.2, Preparation for Certification of Accumulation j and process P6.j, Software Certification for Accumulation j, are expanded into engineering subprocesses (i.e., tasks) for execution by Cleanroom-trained software engineers. These tasks result in the preparation for and performance of the certification of developed software.

The reader of this volume should be familiar with the information and terms described in *Volume 4 - Cleanroom Software Specification.* This volume builds on the theory involving software usage modeling which is introduced in volume 4.

1.1 Background and Motivation

Cleanroom software certification focuses on the principle of Statistical Quality Control (SQC). The goal of SQC is to control and improve product quality by a statistical assessment of the *process* that creates the product. Software testing using SQC allows the projection of the level of correctness that a product will exhibit from testing with only a sample of the total usage of the product. This is a paradigm shift from typical software testing approaches. Traditional software testing attempts to find as many faults as possible before the product is shipped to the customer. The more traditional structural or functional testing approaches can only base their expectation of field behavior of the product on the observation, or lack of observation, of failures from a non-statistical suite of test scenarios. SQC-based software testing allows the inference of the quality observed in the testing environment to the expected quality in the operational environment because testing is conducted from the users' point of view.

Cleanroom follows a notion of SQC similar to that found in the manufacturing industry. Deming's ideas of SQC for manufacturing, conceived in the 1950's, are concerned with imperfect manufacturing of a product in which the statistics are gathered by measuring the quality of a sample of the actual product (each sample item is used once to determine whether or not it works). Mills applied and extended Deming's ideas to software in the 1980's. Mills focuses the concern for software on an imperfect design process in which the statistics are measured from sample uses of a single product (i.e., the software). This makes possible a statistical certification of software based on an operational profile of software usage.

The fact that most software has very large numbers of possible inputs and outputs means that it cannot be exhaustively tested. As a result, any testing approach will exercise only a small portion of the total input domain and output range. Thus the challenge is to find a subset (sample) of inputs that maximize the net gain of testing the software.

Mills argues that statistical testing, in which tests are randomly generated based on a probability distribution that simulates anticipated field usage, is the best strategy for two reasons.

> First, since the test cases represent real usage, statistical testing tends to uncover errors in the order of their contribution to the mean time to failure (*MTTF*) of the software. Thus, as failures are removed, the largest possible gain in the *MTTF* is achieved. Furthermore, any failure likely to be encountered by users is likely to be discovered during the statistical test. If failures remain in the software after a statistical test, they tend to be low probability failures.
>
> Second, the random nature of the test allows the use of rigorous statistical techniques to measure the user-perceived quality of the software. Introducing SQC to software testing allows the testing process to become a process of scientific software certification instead of a seemingly endless cycle of finding and removing failures. In fact, ordinary testing and debugging is an endless cycle indeed because fifteen per cent of the fixes lead to new failures, even with the best of programmers with the best of intentions. This is hard to believe, but it is true! As a result, typical large systems never go to zero failures, but stabilize at a higher level. It is frustrating, but true, that new failures arise, whether the original programmers or new programmers do the fixing. Some failures are due to misconceptions of the original programmers, who imagine a different specification or a different utility than might exist. Some failures are due to new programmers, who imagine a different program design than might exist.

1.2 Cleanroom Process Manual Mission

The mission of this process manual for Cleanroom software certification is to organize and explain the set of specific tasks performed by the Cleanroom Certification Team for each accumulation of a project. Certification tasks are described and the relationships among the tasks are defined. The Cleanroom Certification Team shall create the test plan, test scenarios, and expected execution results for an accumulation. Additionally the Certification Team shall certify the correctness of the accumulation by executing the test scenarios in the manner defined by the test plan, and comparing the actual results to the expected results to determine correctness. Templates are presented as recommended formats for the presentation of information and task results.

1.3 Cleanroom Certification Process Models - P5.j.2, P6.j

The incremental development of the Cleanroom process mandates that the software also be certified an increment at a time. This makes the certification process more manageable and allows the certification and development tasks to proceed in parallel. Additionally, the code for each prior increment is included in the certification of later increments; making the certification for the early increments more thorough. This incremental development allows an iterative assessment of the quality of the code as well as the process that created the code.

The certification results of each increment are independent from the prior increments' certification. None of the data for previous increments is included in the certification of the current increment. However, since the code is included, the final certification (i.e., for the released product) includes pertinent information from each increment by default. The results of the certification of early increments (i.e., those not released to customers) are intended to give feedback to project management about the quality of the process, to developers about the quality of the code, and to certifiers about the effectiveness of the usage model.

As described in Volume 1 of the Cleanroom Process Manuals, Cleanroom system and software development can be described in a detailed process model. A complete development project for a system can be divided into accumulations for certification preparation. The engineering task for the preparation for the certification of accumulation j is labeled P5.j.2, Prepare for Certification of Accumulation j. The detailed process model for P5.j.2 is:

```
proc P5.j.2: Prepare for Certification of Accumulation j
   [This process results in test plan and test scenarios for accumulation j]
   [P5.j.2:  Prepare for Certification of Accumulation j]
   con
      do
         do
               C5.j.2.1:    Prepare test plan;
         until
            Completion Conditions for C5.j.2.1 achieved
         od;
         do
            con
               C5.j.2.2:    Prepare test scenarios or test scenario generator for accumulation j;
               C5.j.2.3:    Determine expected results for test scenarios;
            noc;
         until
            Completion Conditions for C5.j.2.2 and C5.j.2.3 achieved
         od;
      od;
      C5.j.2.5:    Increase Understanding of Problem and Solution Domains;
   noc;
corp;
```

Process P5.j.2, and its component tasks, will be described in detail in section 2 of this document.

As described in Volume 1 of the Cleanroom Process Manuals, Cleanroom system and software development can be described in a detailed process model. A complete development project for a software system can be divided into accumulations for certification. The engineering task for the actual certification of accumulation j is labeled P6.j, Software Certification for Accumulation j. The detailed process model for P6.j is:

```
proc P6.j: Software Certification for Increment j
   [P6.j certifies the code, and makes the decision as to whether an increment will be accepted
   or rejected.]
   do    [P6.j: Software Certification for Increment j]
      C6.j.1:  Build Accumulation j;
      if no pre-certification failures
      then
         while
            certification plan requires more tests and sufficient failures have not been observed
            to make it desirable to terminate testing and wait for corrections
         do
            C6.j.2:  Perform Certification Tests for Accumulation j;
         od;
      fi;
      if at least one failure observed and observed failures are considered to be correctable
      then
         C6.j.3:  Prepare failure report(s);
         D6.j.4:  Correct failure, verify correction and prepare ECN;
      fi;
      M6.j.5:  Management Decision: (1) certification complete-accumulation quality satisfactory,
               (2) certification complete-quality not satisfactory-replanning Is required or (3)
               certification should continue;
      if not Management Decision is certification should continue
      then
         C6.j.6:  Prepare Certification Report;
      fi;
   until
      Completion Conditions achieved for C6.j.6
   od;
corp;
```

Process P6.j, and its subtasks, will be described in detail in section 3 of this document.

SECTION 2: DEVELOP AND ANALYZE MARKOV USAGE MODEL TASKS

The material in this section directly supports the work of the Specification Team in the preparation of the Volume V of the specifications. The engineering task to Develop and Analyze the Markov Usage Model is S4.i.2.7 and the engineering task to prepare Volume V of the specification is S4.i.4.5 Record Cycle i Results in Usage Profile Volume. This material is being discussed in this volume because of the logical connection preparing the usage profile and then using the profile to support the software certification.

Engineering task S4.i.2.7 can be decomposed into finer engineering tasks as follows:

```
proc S4.i.2.7: Develop and Analyze Markov Usage Model
   do    [S4.i.2.7: Develop and Analyze Markov Usage Model]
      con
         S4.i.2.7.1:  Determine Usage States;
         S4.i.2.7.2:  Determine Transition Possibilities and Transition Stimuli;
         S4.i.2.7.3:  Determine User Classes That Need To Be Modeled;
         for each class of user to be modeled
         do
            S4.i.2.7.4:  Determine Transition Probabilities;
            S4.1.2.7.5:  Analyze Usage Model;
         od;
         S4.i.2.7.6:  Document Model(s) In Specification Volume V;
      noc;
   until
      usage model(s) and specification volume are complete
   od;
corp;
```

Engineering practices for performing each of these engineering tasks are discussed in the in this section. The result of this work is documented in volume 5 of the Specification.

2.1 Usage Profile

A *usage profile* for a software system or any accumulation of a Cleanroom project is an external description of software interaction with users. Users are any human, program, device, etc. that can apply stimuli to the software. A usage profile is defined by a Markov model consisting of a set of states which represent the operational states (called the *usage state*) of the software and a set of arcs labeled with external stimuli which cause state transitions. Associated with each state is a stimulus distribution that simulates the application of stimuli from that state by a particular user. A usage profile defined in this manner is a formal mathematical system; namely,

a discrete parameter Markov chain. This *usage Markov chain* is a source for analysis that lends insight into important usage characteristics of the software.

The objective of this section is to specify engineering practices that are helpful in developing a usage profile model. Usage models can be specified in either a graphical form or a table. A tabular form is specified in Table 2.1.1.

Table 2.1.1 Tabular Format For Usage Model for One Class Of User

From State	Transition Stimuli	To State	Transition Probability

Graphical displays are useful for studying some aspects of the model during its preparation or to help explain the model. A typical graphical display would be a state transition diagram as shown in Figure 2.3.1 (this figure is explain in detail later) with the addition of arc labels to define transition probabilities.

A certification tool called the Certification Assistant (CA) has been developed by SET, Inc. which processes the above tabular format and automates much of the analysis described in this process manual. Aspects of the tool are described in various sections of this volume.

2.2 Determine Usage States (S4.i.2.7.1)

As the software operates, it moves from one operational state to another operational state. The first step in developing a usage profile is to specify all the operational states. This list of states can be developed by an examination of Specification volumes II and III. Operational states have nothing to do with actual implementation. These are the states defined by the understanding of the problem and solution domains of the system to be implemented.

Consider the simple selection menu pictured in Figure 2.2.1. The input domain and simplified transition rules appear in Table 2.2.1. The first menu item, Project Manager, is used to define a project (the semantics are not described here for simplicity). The project name then appears in the upper right corner of the screen. Once a project is defined the next three items, Network Definition, Reliability Information, and Certification Model can be selected to perform their respective functions (also not described). If no project is defined, these items cannot be selected.

Figure 2.2.1: An Example Screen Layout

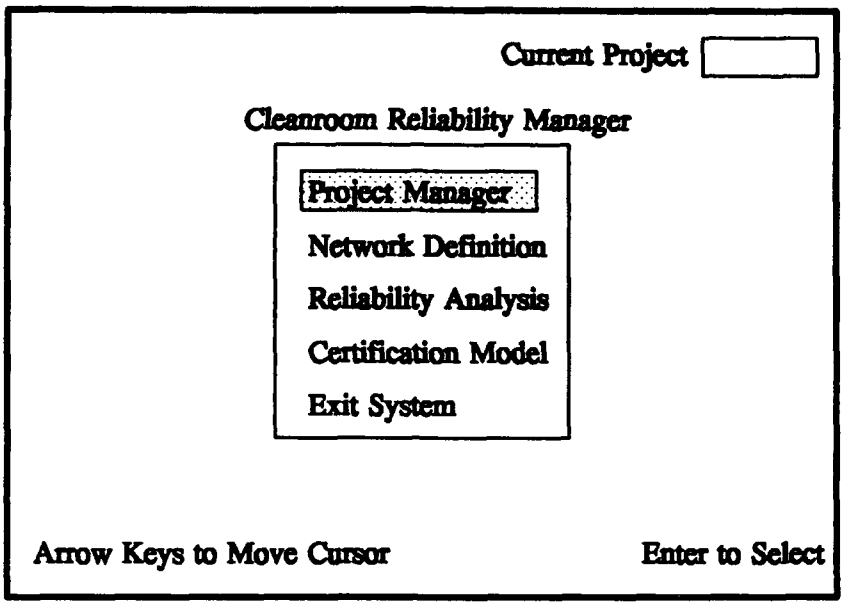


Table 2.2.1 - A Simplified Specification for the Example Menu

Stimulus	Condition	Response
Invoke		Display menu with "Project Manager" highlighted
Invoke w/filename	filename is not a valid project	Display menu with "Project Manager" highlighted
	filename is a valid project	Display menu with "Project Manager" highlighted and filename appearing in the Current Project box
Up Arrow key (↑)	"Project Manager" highlighted	Move highlight bar to "Exit System"
	otherwise	Move highlight bar up one line
Down Arrow key (↓)	"Exit System" highlighted	Move highlight bar to "Project Manager"
	otherwise	Move highlight bar down one line

Stimulus	Condition	Response
Enter key (↵)	"Project Manager" highlighted	Display the Project Manager screen
	"Exit System" highlighted	Terminate the software
	highlight bar on any other option and no active project	null
	highlight bar on any other option and a project is active	Display the appropriate screen
Any other key		null

Determining the usage states for the software requires creation and design similar to creating a state box from a black box. There is no algorithm that can accomplish this task; however, one must keep in mind that usage states are determined from a users' point of view. Examples of common usage states for applications software are: cursor location, window configuration, and data file attributes. For real-time software, common usage states are: bus or device phase, buffer contents, instrument or sensor readings, exception conditions, and timing considerations.

For this example, usage states are defined to be those attributes of the software affecting the application of stimuli by the user. In this example, the state is (i) the current screen (menu=S1, project manager=S2, network definition=S3, reliability information=S4, and certification model=S5) , (ii) the location of the highlighted bar (which for screen 1 is: project manager=L1, network definition=L2, reliability information=L3, certification model=L4, and exit system=L5), and (iii) whether or not a project is active (Yes or No). Thus a usage state appears as a triple; for example, (S1,L3,Yes) is an element of the usage state set.

Invocation of the software can occur with either no filename supplied, an illegal filename, or a legal filename. In the case of the first two, the usage state becomes (S1,L1,No), i.e., the first screen is displayed, the highlight is on option 1, and no project is currently active. In the case of invocation with a legal filename, the usage state is (S1,L1,Yes). From each of these new states, all possible stimuli must be considered. These stimuli are: the arrow keys, which change the location of the highlight bar, and the enter key, which either causes a null response when no project is active or a screen change when a project is active.

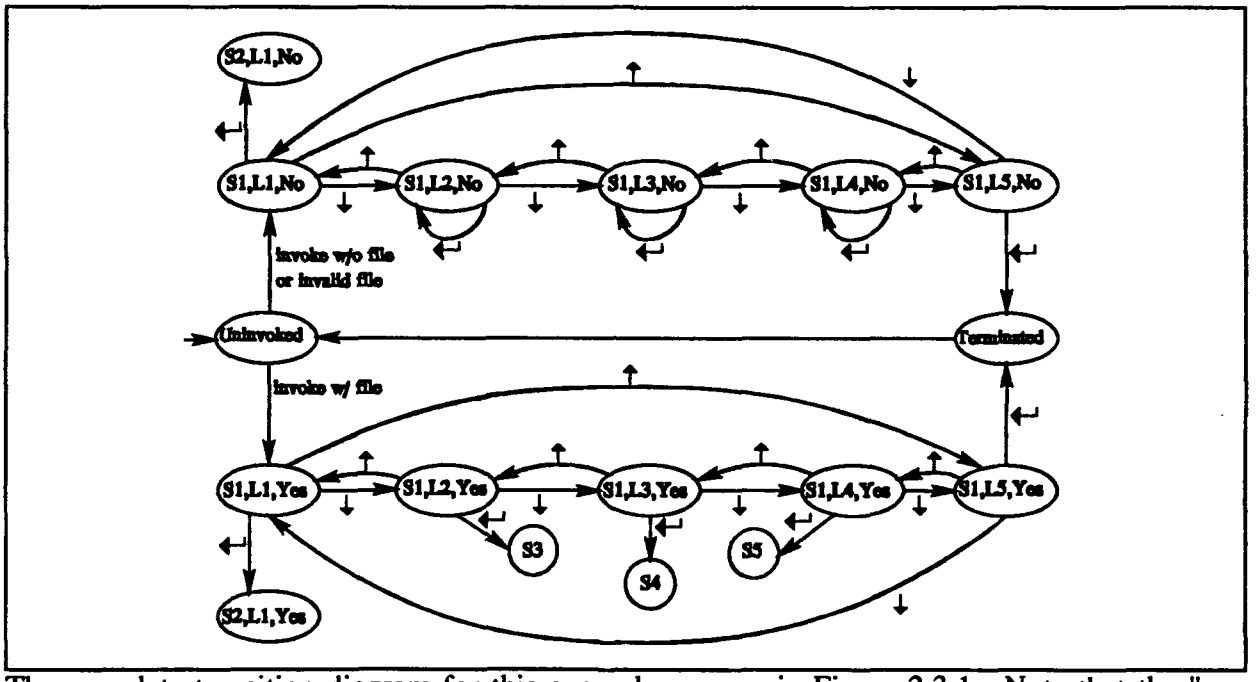
This information is recorded in Section 1 of Volume V of the specification.

2.3 Determine Transition Possibilities and Transition Stimuli (S4.i.2.7.2)

In order to describe the transitions between system states, it is necessary to understand the nature of the stimuli to the system. The potential stimuli for each state need to be listed.

The usage states and stimuli can be organized as the expected usage profile. The format of a usage profile is a state transition diagram (STD). The STD is organized by connecting usage states with directed arcs, labeled with appropriate stimuli (and their probability), which cause the state transitions. Alternatively, a transition matrix is an equivalent representation using the states as indices and the transitions (stimuli) as the matrix entries.

Figure 2.3.1: An Example State Transition Diagram



The complete transition diagram for this example appears in Figure 2.3.1. Note that the "any other key" stimulus could be included by installing an arc from each state to itself labeled appropriately. As with all aspects of Cleanroom, the transition diagram can be represented in either a graphical (STD) or a tabular (text) format.

2.4 Determine User Classes (S4.i.2.7.3)

Certification requirements typically include multiple user types that are foreseen as potential end-users. Each distinct class of user should be listed. Additionally, some applications have distinct *modes* or usage. For example, a software system may have an installation mode and a mode of normal operation. Each such mode should also be listed.

Each user class and set of requirements may necessitate the definition of a *stratum of use*. A stratum is a subset of the usage profile that describes a particular user type or satisfies a requirement or set of requirements. For each stratum, several questions must be answered. First, where will the randomization will occur in the stratum. Randomization can be in either the control flow, the data items, or (preferably) both. Second, statistical aspects of the stratum must be determined through analysis. The analysis will describe pertinent statistical properties of the stratum including, expected number of stimuli/states in a scenario and the expected makeup of each scenario. Third, for each stratum, the mapping back to user classes and requirements must be made explicit. The purpose of each stratum and the reason for including it in the certification needs to be clear.

Each stratum will have an associated usage profile.

2.5 Determine Transition Probabilities (S4.i.2.7.4)

The usage profile is a mechanism that is capable of randomizing both the control flow through the software as well as the data that is input to the software. For example, if the user can enter an alphanumeric string then the usage profile should allow for all possible combinations of alphanumeric characters that comprise such a string, i.e., a string of random length with random values. Thus the stimuli distributions of the usage model take the form of either a single symbol of alphanumeric data or a series of symbols. Of course the sensible values must be given a higher probability weight than the totally random ones. The goal is to randomize every aspect of use (control flow and data) but the cost of doing this may be too great in some instances. If it is determined that it is too costly to randomize all the data, certifiers can either sample from actual data or pre-generate random data. States and arcs are installed in the model to instruct the certifier to select from this pre-generated data when it is needed.

Transition probabilities can be established in three different ways.

1. If actual user sequences are available from a prior version or prototype of the software, stimuli transitions in these sequences can be counted to obtain relative frequency estimates of the transition probabilities. This is the most desirable method of establishing transition probabilities.

   The transition probabilities are established by counting corresponding stimuli in the user sequences and converting these to probabilities (dividing each frequency by the sum of the frequencies of all arcs that leave a given state). One must take care to obtain sequences that are *structurally complete*, i.e., each transition defined in the chain appears in the user sequences. If structurally complete sequences are not available, a good practice is to initialize each frequency count to 1 before counting the frequencies in the sequences.

2. One can use hypothesized sequences of use to determine relative frequency probabilities if there is some knowledge of intended (or even sensible) use. In the absence of explicit usage information, this is often the best way to obtain the transition probabilities.

   Note that the issue of structural completeness must be resolved in the case of hypothesized sequences as well.

   Hypothesizing sequences of use requires that the software engineers place themselves in the role of the user and construct sequences of use (assuming that the user will use the system in a careful and reasonable manner) to perform each black box subfunction. For example, the menu example used above has five options from which the user can choose, thus, sequences that describe all the "sensible" ways of executing each option can be constructed. These sequences are listed below.

   To select the PROJECT MANAGER option,

   (i) Invoke w/o file, Enter
   (ii) Invoke w/ illegal file, Enter

   To select the NETWORK DEFINITION option

   (iii) Invoke w/ file, Dn Arrow, Enter

   To select the RELIABILITY INFORMATION option

   (iv) Invoke w/ file, Dn Arrow, Dn Arrow, Enter

   To select the CERTIFICATION MODEL option

   (v) Invoke w/ file, Dn Arrow, Dn Arrow, Dn Arrow, Enter
   (vi) Invoke w/ file, Up Arrow Up Arrow, Enter

   To select the EXIT SYSTEM option

   (vii) Invoke w/ file, Dn Arrow, Dn Arrow, Dn Arrow, Dn Arrow, Enter
   (viii) Invoke w/ file, Up Arrow, Enter

   The frequency counts (and, therefore, probabilities) can be tabulated using the format of Table 2.1.1 as follows. Each arc is assigned an initial frequency of 1. Now each of the above sequences are used to increment the frequency count on the corresponding arc to obtain informed probabilities. Table 2.5.1 lists each frequency and probability obtained in this manner.

Table 2.5.1 Obtaining Informed Probabilities From Hypothesized Sequences

From State	Transition Stimuli	To State	Transition Probability
Uninvoked {-,-,-}	invoke w/o filename or invoke w/invalid file	{S1,L1,No}	3/10
	invoke w/valid file	{S1,L1,Yes}	7/10
{S1,L1,No}	↓	{S1,L2,No}	1/5
	↑	{S1,L5,No}	1/5
	↵	{S2,L1,No}	3/5
{S1,L2,No}	↓	{S1,L3,No}	1/3
	↑	{S1,L1,No}	1/3
	↵	{S1,L2,No}	1/3
{S1,L3,No}	↓	{S1,L4,No}	1/3
	↑	{S1,L2,No}	1/3
	↵	{S1,L3,No}	1/3
{S1,L4,No}	↓	{S1,L5,No}	1/3
	↑	{S1,L3,No}	1/3
	↵	{S1,L4,No}	1/3
{S1,L5,No}	↓	{S1,L1,No}	1/3
	↑	{S1,L4,No}	1/3
	↵	Terminated	1/3
{S1,L1,Yes}	↓	{S1,L2,Yes}	5/9
	↑	{S1,L5,Yes}	3/9
	↵	{S2,L1,Yes}	1/9

From State	Transition Stimuli	To State	Transition Probability
{S1,L2,Yes}	↓	{S1,L3,Yes}	4/7
	↑	{S1,L1,Yes}	1/7
	↵	{S1,L2,Yes}	2/7
{S1,L3,Yes}	↓	{S1,L4,Yes}	3/6
	↑	{S1,L2,Yes}	1/6
	↵	{S1,L3,Yes}	2/6
{S1,L4,Yes}	↓	{S1,L5,Yes}	2/6
	↑	{S1,L3,Yes}	1/6
	↵	{S1,L4,Yes}	3/6
{S1,L5,Yes}	↓	{S1,L1,No}	1/6
	↑	{S1,L4,No}	2/6
	↵	Terminated	3/6
{S2,L1,No}	null	{S1,L1,No}	1
{S2,L1,Yes}	null	{S1,L1,Yes}	1
S3	null	{S1,L2,Yes}	1
S4	null	{S1,L3,Yes}	1
S5	null	{S1,L4,Yes}	1
Terminated	null	Uninvoked	1

3. In the absence of any information whatsoever, a "maximum uncertainty" estimate can be used by assigning a uniform probability distribution across the exit arcs at each state. Despite the fact that this is a completely uninformed method of assigning transition probabilities, it can result in an effective statistical test. Often, the analytical results, discussed in the next section, give insight into minor adjustments that increase the usefulness of the model.

The first two approaches lead to informed probabilities. The first approach will generate the most accurate estimate of probabilities but in many case the cost may be judged to be greater than the

benefit or the alternative systems are not available. The second approach can always be used. The third approach leads to uninformed probabilities.

2.6 Analyze Usage Models (S4.i.2.7.5)

A complete usage Markov chain is a formal and mathematical system and is therefore subject to comprehensive statistical analysis. The analysis is aimed at answering question such as:

- How long will it take a user to supply each stimulus?
- With what distribution are the stimuli being supplied by the users?
- With what distribution are the sequences being supplied by the users?
- Which parts of the system will see the most use?
- If the transition probabilities are changed, what is the net effect on system usage?

The analysis takes the form of defining random variables that describe software usage and then deriving their expectation and standard deviation directly from the Markov chain. The following is a list of random variables which have been identified, derived, and interpreted and are available for immediate incorporation into any usage modeling application.

- the number of steps until a given stimulus appears
- the number of steps until all stimuli appear
- the probability with which each stimulus appears over time
- the number of steps until a stimulus recurs (appears again)
- the number of appearances of a stimulus in a single user session
- the number of user sessions between appearances of a given stimulus
- the probability that a given stimulus appears in a session
- the number of sessions until a given stimulus appears
- the number of sessions until all stimuli appear
- the probability that a given session will occur

A sample of analytical results for the example menu appear in Table 2.6.1 using the hypothesized data for stimulus distributions and in Table 2.6.2 using uninformed probabilities. These results are computed automatically by SET's CA software. The CA tool takes as input the tabular format of the usage profile and creates usage statistics reports which are helpful in obtaining accurate usage representation and in optimizing the effectiveness of the statistical test.

The analytical results allow engineers to discover the properties and capabilities of the usage model that are not obvious from the model itself. The best example of this is the expected time until all the stimuli appear. This is in no way obvious by looking at the model, but it is a very important measure for developers in determining which stimuli appear more often than others and for certifiers when they are attempting to determine an expected test budget.

The Specification Team must determine whether or not these results match any known usage information or intuition. If so, then the usage profile developed is a suitable starting point for the Certification Team. If not, then either structural (states and arcs) changes or statistical (transition probabilities) changes must be made and the results recomputed and analyzed to determine whether a more appropriate profile has been achieved. In other words, the construction of the usage model and the estimation of its parameters is an iterative process that involves several cycles of estimation and analysis before an appropriate model is obtained.

If changes to the model are necessary, the analytical results are used to indicate where the adjustments can most effectively be made. For example, if a stimulus has too low a probability, one should adjust the transition probabilities corresponding to the arcs on which the stimulus appears and/or the arcs leading to the stimulus from the invocation state.

Table 2.6.1 - Analytical Results for the Usage Chain with Informed Probabilities

The expected use length: 21.7 stimuli
The expected number of uses until each state is visited: 8.43
The expected number of uses until each arc is traversed: 16.67

A: The expected number of state transitions until each state occurs from Uninvoked
(and standard deviation)
B: The probability that a state occurs between Uninvoked and Terminated
C: The expected number of occurrences of a state between Uninvoked and Terminated
(and standard deviation)
D: The steady state probability of each state
E: The expected time until a state recurs
(and standard deviation)

State {screen,line,filename}	A	B	C	D	E
Uninvoked {-,-,-}	22.73 (21.28)	1.0	1.0 (0)	0.044010	22.73 (22.01)
{S1,L1,No}	44.74 (57.38)	0.3	2.7 (6.21)	0.118827	8.42 (7.96)
{S1,L2,No}	71.14 (77.78)	0.22	1.44 (3.94)	0.063375	15.80 (15.45)
{S1,L3,No}	87.02 (89.90)	0.19	1.26 (3.71)	0.055453	18.05 (17.61)
{S1,L4,No}	85.61 (86.71)	0.2	1.08 (3.07)	0.047531	21.05 (19.87)
{S1,L5,No}	54.74 (58.25)	0.3	0.9 (1.92)	0.039609	25.25 (21.75)

State {screen,line,filename}	A	B	C	D	E
{S1,L1,Yes}	14.71 (31.33)	0.7	1.28 (1.33)	0.056550	17.70 (17.30)
{S1,L2,Yes}	25.32 (40.79)	0.5	1.46 (2.24)	0.064334	15.55 (14.76)
{S1,L3,Yes}	27.59 (40.90)	0.49	1.98 (3.19)	0.087216	11.47 (10.98)
{S1,L4,Yes}	26.89 (38.22)	0.53	2.92 (4.53)	0.128292	7.80 (5.43)
{S1,L5,Yes}	22.73 (32.37)	0.7	1.4 (1.5)	0.061614	16.23 (15.45)
{S2,L1,No}	57.77 (67.51)	0.25	1.62 (4.06)	0.071296	14.04 (12.94)
{S2,L1,Yes}	172.87 (187.02)	0.12	0.14 (0.42)	0.006283	161.29 (161.13)
{S3}	78.72 (90.59)	0.23	0.42 (0.97)	0.018381	54.64 (50.87)
{S4}	60.99 (70.27)	0.28	0.66 (1.42)	0.029072	34.48 (34.29)
{S5}	41.48 (50.31)	0.39	1.46 (2.71)	0.064146	15.60 (15.43)
Terminated {-,-,-}	21.73 (21.28)	1.0	1.0 (0)	0.044010	22.73 (22.01)

Table 2.6.2 - Analytical Results for the Usage Chain with Uninformed Probabilities

The expected use length: 25.9 stimuli
The expected number of uses until each state is visited: 4.67
The expected number of uses until each arc is traversed: 9.43

A: The expected number of state transitions until each state occurs from Uninvoked
B: The probability that a state occurs between Uninvoked and Terminated
C: The expected number of occurrences of a state between Uninvoked and Terminated
D: The steady state probability of each state
E: The expected time until a state recurs

State {screen,line,filename}	A	B	C	D	E
Uninvoked {-,-,-}	26.9 (25.25)	1.0	1.0 (0)	0.037175	26.90 (23.65)
{S1,L1,No}	30 (49.44)	0.5	2.7 (4.38)	0.100372	9.96 (6.52)
{S1,L2,No}	47.38 (63.68)	0.36	2.4 (4.85)	0.089219	11.21 (7.45)
{S1,L3,No}	58.14 (71.66)	0.32	2.1 (4.61)	0.078067	12.81 (8.96)
{S1,L4,No}	57.5 (69.41)	0.33	1.8 (3.79)	0.066914	14.94 (10.82)
{S1,L5,No}	36.8 (50.01)	0.5	1.5 (2.29)	0.055762	17.93 (11.36)
{S1,L1,Yes}	25.8 (41.65)	0.5	2.7 (4.38)	0.100372	9.96 (6.52)
{S1,L2,Yes}	41.97 (54.32)	0.36	2.4 (4.85)	0.089219	11.21 (7.45)
{S1,L3,Yes}	52.54 (61.54)	0.32	2.1 (4.61)	0.078067	12.81 (8.96)
{S1,L4,Yes}	52.7 (59.87)	0.33	1.8 (3.79)	0.066914	14.94 (10.82)
{S1,L5,Yes}	33.8 (42.74)	0.5	1.5 (2.29)	0.055762	17.93 (11.36)
{S2,L1,No}	58.89 (73.48)	0.32	0.9 (1.82)	0.033457	29.89 (20.73)
{S2,L1,Yes}	54.69 (65.12)	0.32	0.9 (1.82)	0.033457	29.89 (20.73)
{S3}	74.6 (81.01)	0.25	0.8 (1.92)	0.029740	33.62 (26.14)
{S4}	89.97 (92.98)	0.22	0.7 (1.81)	0.026022	38.43 (31.48)

State {screen,line,filename}	A	B	C	D	E
{S4}	89.97 (92.98)	0.22	0.7 (1.81)	0.026022	38.43 (31.48)
{S5}	96.53 (97.75)	0.21	0.6 (1.55)	0.022305	44.83 (34.25)
Terminated {-,-,-}	25.9 (25.25)	1.0	1.0 (0)	0.037175	26.90 (23.65)

Understanding and interpreting the analytical results is an important step for the certification team. The Certification Assistant tool will compute the results automatically and print summary reports as well in order to aid in this analysis process. The analytical results are also packaged into a coverage report by the CA which gives a return-on-investment analysis. The report gives a sequence-by-sequence analysis of the expected coverage of states and arcs of the usage model. The report is shown in Table 2.6.3.

Table 2.6.3 Expected Coverage Report

Sequence Number	% States Covered	% Arcs Covered
1	11.76	87.20
2	29.41	87.54
3	47.06	88.58
4	70.59	90.31
5	88.24	91.70
6	94.12	92.39
7	94.12	92.73
8	94.12	93.08
9	100.00	93.77
10	100.00	94.81
11	100.00	94.81
12	100.00	95.16
13	100.00	95.85
14	100.00	96.89
15	100.00	98.27
16	100.00	99.31
17	100.00	100.00

2.7 Document Model(s) In Specification (S4.i.2.7.6)

The models and all supporting analyses needs to be documented in Volume V of the Specification. The outline specified in Section 5.5 of Volume 4 can be used to guide the preparation of the volume. The outline for that volume is:

Section 1: Usage States

This section contains a list of each usage state element and an enumeration of each usage state. If any usage states are omitted, the reasoning supporting the omission should be included in this section.

Section 2: Transition Possibilities and Transition Stimuli

This section should contain either the STD and/or the tabular representation of the transitions among the usage states, the stimuli that cause each transition, and the probability with which each transition occurs. Furthermore, some reasoning of each distribution should be included. If any real or hypothesized data was used to determine the probabilities, it should be included or referenced.

Section 3: User Classes

Each user class should be listed along with justification of its importance.

Section 4: Usage Model Analysis

The statistical analysis of section 2.6 should be performed in its entirety and included in the specification. Any results of specific interest should be highlighted. Specifically, any resulting arguments about the appropriateness of the probabilities should be included that is based on the analytical results.

Section 5: Assumptions

In developing the usage profile, it may be necessary to make certain assumptions. For example, a description of assumptions concerning the usage states and stimuli distributions should be completely documented to the satisfaction of all appropriate stakeholders.

2.8 Usage Profiles For Increments

The software will be constructed and certified in increments. Each increment is executable by user commands. In order to measure the reliability of each increment, it is necessary to develop a usage profile for each increment from the usage profile for the entire system. The expected usage profiles for each increment are developed in process P5.j.1. The Markov model for the entire system is specialized for each accumulation of increments. Procedures for doing this are discussed in section 3.2.

SECTION 3: PREPARE FOR CERTIFICATION OF ACCUMULATION J TASKS

Both processes P5.j.1 and P5.j.2 include engineering tasks that prepare for the certification of an accumulation. Process P5.j.1 includes engineering task S5.j.1.2 which results in a usage profile tailored to accumulation j.

```
proc P5.j.1:   Tailor Specification to Increment/Accumulation j
  do [P5.j.1: Tailor Specification to Increment/Accumulation j]
    con
      S5.j.1.1:   Tailor Black Box function to increment/accumulation j;
      S5.j.1.2:   Tailor Usage Profile to increment/accumulation j;
    noc;
  until
    Completion Conditions achieved for S5.j.1.1 and S5.j.1.2
  od;
corp;
```

Engineering practices for performing task S5.j.1.2 are discussed in section 3.1.

Process P5.j.2 includes all the other tasks that are required to prepare for the certification of an accumulation.

```
proc P5.j.2: Prepare for Certification of Accumulation j
  [This process results in test plan and test scenarios for accumulation j]
  [P5.j.2:  Prepare for Certification of Accumulation j]
  con
    do
      do
        C5.j.2.1:   Prepare Test Plan;
      until
        Completion Conditions for C5.j.2.1 Achieved
      od;
      do
        con
          C5.j.2.2:   Prepare Test Scenarios or Test Scenario Generator for Accumulation
                      j;
          C5.j.2.3:   Determine Expected Test Results;
        noc;
      until
        Completion Conditions for C5.j.2.2 and C5.j.2.3 Achieved
      od;
    od;
```

C5.j.2.4: Increase Understanding of Problem and Solution Domains;
noc;
corp;

Process P5.j.2, Preparation for Certification of Accumulation j, consists of four tasks (i) prepare test plan, (ii) prepare test scenarios or test scenario generator for accumulation j, (iii) determine expected results for test scenarios, and (iv) increase understanding of problem and solution domains. These tasks result in the accumulation test plan deliverable and put the Certification Team in position to certify an accumulation of a software system. Engineering practices for performing these four tasks are discussed in sections 3.2 through 3.5.

These activities support the creation of documentation necessary to certify an accumulation of a software system. The Completion Conditions for each task can be found in Volume 3.

3.1 Tailor Usage Profile to Accumulation j (S5.j.1.2)

A subset of the usage profiles for each user class is developed from the specification and included in Volume 5 of the specification volume. A subset of the usage profile must be developed for each accumulation except the last accumulation whose profile is equal to the entire product profile.

The usage profile developed during the specification phase is specialized to meet the goals specified in the test plan. Specifically, the *Markov chains*[1] corresponding to each stratum are constructed and analyzed.

The Markov chains for the strata are a subsets of the STD for the accumulation.

A stratum Markov chain can sometimes be directly extracted from the accumulation Markov chain simply by removing states and/or stimuli and normalizing the probabilities. This creates a transition matrix which is a subset of the usage profile. It is acceptable to introduce bookkeeping states and/or null stimuli in order to simplify the sub-models and maintain consistent organization of the sub-models.

Before any testing begins, the accumulation model should be analyzed as described in section 2.6. The following analytical results are available from the analysis:

- the number of steps until a given stimulus appears in a test scenario
- the number of steps until *all* stimuli appear in a test scenario
- the probability with which each stimulus appears over time
- the number of steps until a stimulus recurs (appears again)

---

[1]The STD with probabilities attached to the arcs describes a finite-state, discrete-parameter Markov process. These processes are commonly called Markov chains. In this process manual, one can think of the Markov chain as either the probabilistic STD or as a transition matrix with the states as indices and the transition probabilities as entries.

- the number of appearances of a stimulus in a single test scenario
- the number of sequences that appear between appearances of a given stimulus
- the probability that a given stimulus appears in a sequence
- the number of sequences until a given stimulus appears
- the number of sequences until *all* stimuli appear

This information is used to compare the behavior of the stratum Markov chains with the expected usage results in the test plan. Any inconsistencies must be removed before test scenarios are prepared.

The analytical results give information that needs to be included in the test plan; specifically, the information that gives insight into budget considerations (the expected amount of testing until certain events occur, etc.) should be included in the section on testing constraints/level of effort.

3.2 Prepare Test Plan (C5.j.2.1)

Preparing an adequate test plan for an accumulation of a project is the first step toward accurate software certification. This test plan is an extension and augmentation of the usage profile. During the specification phase, the principal objective is to describe usage of the software and to gain an understanding of how this usage will affect the development and testing of the software product. During the certification planning phase, the usage profile is augmented by including other testing-related issues such as requirements modeling, mission-critical usage, stratification, etc. The test plan is created during this phase and describes in detail the results achieved during the planning process. The test plan has the following sections:

1. Certification Statement
2. Testing Requirements
3. Testing Constraints
4. Definition of Strata

Test Plan For [Accumulation j]

Section 1: Certification Statement

The first step in developing an adequate test plan is to organize a certification statement for the entire software product. That is, what information should the certification process give stakeholders about operation of the software? Once this statement is created and agreed upon by all stakeholders, the usage profile is specialized so that the certification statement can be fulfilled.

The certification statement should cover all important aspects of a software product. The first such aspect is the intended quality level of the certified product and each accumulation. An example is as follows.

The accumulation must show a reliability level of not less than x% for each version[2] of the accumulation in the statistical test.

If the accumulation will be released to users then the statement may be altered as follows.

The accumulation must show a reliability level of not less than x% for each intermediate version in the statistical test and the final version must exhibit failure-free (i.e., correct) behavior.

Such statements will aid the Certification Team in making stopping decisions and guide them in determining when to submit failure reports to (and receive engineering change notices from) the Development Team.

Another important aspect of test planning is to ensure that the certification is meaningful. This requires accounting for each important attribute of the accumulation in the test scenarios so that the certification report (which is based on the scenarios) is worthwhile to the end-users. For example, if one claims a 99% confidence in the correctness of their software but fails to address any safety-critical issues in the statistical test, then the correctness measure is meaningless to a large number of stakeholders. Likewise, if the statistical test fails to cover important test requirements that are mandated by certain stakeholders, then the certification, no matter how accurate, is incomplete. Thus, the certification statement may be again altered as follows.

The accumulation must show a reliability level of not less than x% for each intermediate version in the statistical test and the final version must exhibit failure-free (i.e., correct) behavior. Furthermore, the requirements: r_1, r_2, ... r_m must each be addressed during the test.

Section 2: Testing Requirements

It is crucial that all testing requirements be stated in test planning phase so that they are addressed during the statistical test (and thus included in the certification). These include: (i) all requirements of interest to stakeholders; including any safety-critical or mission-critical functions that the software performs and (ii) attributes of the input domain and/or internal software state that should be observed during the test. It is important to remember that the certification results are conditioned on the usage profile. Any important attribute of the software is certified by including it in the usage profile and mandating its appearance in the statistical test (via the certification statement). Of course, one could simply craft a few well-designed test cases in order to cover important events; however, this keeps such information from being included in the certification. We prefer including as much information as possible into the usage profile so that all knowledge about the software's ability to perform its intended function is embodied in the certification results.

---

[2]A new version of an accumulation is obtained whenever changes to the software occur as a result of errors located during the statistical test.

Section 3: Testing Constraints

Any testing constraints such as maximum allowable expenditure of resources should be included in the test plan. Of particular importance are personnel resources and time constraints because these will impact the number of strata that one may be limited to in the certification.

Section 4: Definition of Strata

Each user class and set of requirements may necessitate the definition of a *stratum of use.* A stratum is a subset of the usage profile that describes a particular user type or satisfies a requirement or set of requirements. For each stratum, several questions must be answered. First, where will the randomization will occur in the stratum. Randomization can be in either the control flow, the data items, or (preferably) both. Second, statistical aspects of the stratum must be determined through analysis. The analysis will describe pertinent statistical properties of the stratum including, expected number of stimuli/states in a scenario and the expected makeup of each scenario. Third, for each stratum, the mapping back to user classes and requirements must be made explicit. The purpose of each stratum and the reason for including it in the certification needs to be clear.

Some stratum may or may not be the focus of the intermediate increments of a software product. In fact it is reasonable to choose different stratum for each intermediate increment to increase test diversity and then certify the final increment using all the strata. If such a strategy is to be used, each stratum should be tied to one or more increments and documented in this section.

For each stratum in an accumulation, the following items are required in the test plan.

1. Stratum Description and Justification
2. Randomization Rules
3. Definition of a Use
4. Identification of Supporting Data Files
5. Usage Analysis Expectations
6. Constraints/Level of Effort

Section 4.1: Stratum Description and Justification

The goal of defining strata for an accumulation is to optimally characterize usage of the software. A stratum for each user type is possible as well as any additional strata that increase the value of the certification for the stakeholders. In particular, the entire accumulation usage profile represents the most general stratuM for an accumulation. Other useful strata are (i) an illegal usage stratum, which models incorrect stimuli distributions, (ii) a critical function stratum, which models exception conditions and other low probability events that may not occur under normal usage, and (iii) a quickly convergent stratum, which allows for fast input domain coverage. Multiple user profiles require that the appropriate states and arcs for the increment under test be duplicated and that new transition probabilities be established that characterize the new stratum.

These profiles would represent either separate test beds yielding multiple certifications of the software or can be integrated into a single model which represents a combined user profile (an average across all users and all strata). The costs of multiple tests must be weighed against the additional information such test provide in order to determine whether such strata are desirable. Multiple models raise both the cost and the effectiveness of certification. The cost might be offset somewhat by manipulating the transition probabilities of the strata profiles to obtain faster convergence.

Each stratum is justified by mapping the need for the stratum to a specific user class or testing requirement. The coverage of the testing requirements and user classes is documented by creating this mapping.

Section 4.2: Randomization Rules

For each stratum, a decision needs to be made about the nature of the randomization present in the test generation process. For example, one may want to randomize the control flow through the software during the test but use a fixed set of data files as input. The opposite could also be true, randomly generated data files could be used during the execution of specific control flow sequences. In general, one should attempt to randomize as many aspects of the test as possible.

In the case that a defined stratum has only a finite number of test scenarios associated with it, statistical testing can (and should) be replaced by exhaustive testing. Such an exhaustive test serves as a proof of correctness given the stratum.

Section 4.3: Definition of a Use

The planning phase is not complete until an acceptable definition of a *use* for each stratum is established. This definition will likely be some suitable description of a single user-session with the accumulation. For example, a user session that begins with invocation of the software and ends with termination of the software is a common definition of a use for interface-driven systems. In a real-time environment it might be power-up to power-down or logon to logoff. It is a good idea to include some sample sequences of use once this definition is established and agreed upon by all stakeholders.

Certification establishes a quality measure based on the definition of a use, i.e., the software performs correctly on *x%* of uses. The length of a use can be measured in numbers of stimuli, processor time, wall clock time, etc. A certification goal establishes the target criterion for the statistical test. This goal can be in terms of reliability (e.g., .95, .98, 2 sigma, etc.) or *MTTF* (e.g., 1000 uses, etc.) whichever criterion makes the most sense for the application. In any case, testing continues until the target measurement, specified in the certification statement, is reached.

Section 4.4: Identification of Supporting Data Files

The file identification of the state transition diagram (STD) files that are input to any automated processing should be included for easy reference. Additionally, any supporting documentation should be identified.

Section 4.5: Usage Analysis Expectations

The analysis presented in this volume, section 2.4, is used to evaluate the stratum sub-model and any specific profiles for the accumulation. This analysis will give insight into the testing considerations for the specific accumulation. Any knowledge or intuition about usage patterns of each stratum should be documented so that it can be compared to the results of the statistical analysis.

Section 4.6: Constraints/Level of Effort

The testing constraints (i.e., resources, budget, etc.) and level of effort that affect each stratum is documented so that management decisions can be made to resolve any conflicts.

The test planning phase is complete when the certification statement is agreed upon by all appropriate stakeholders.

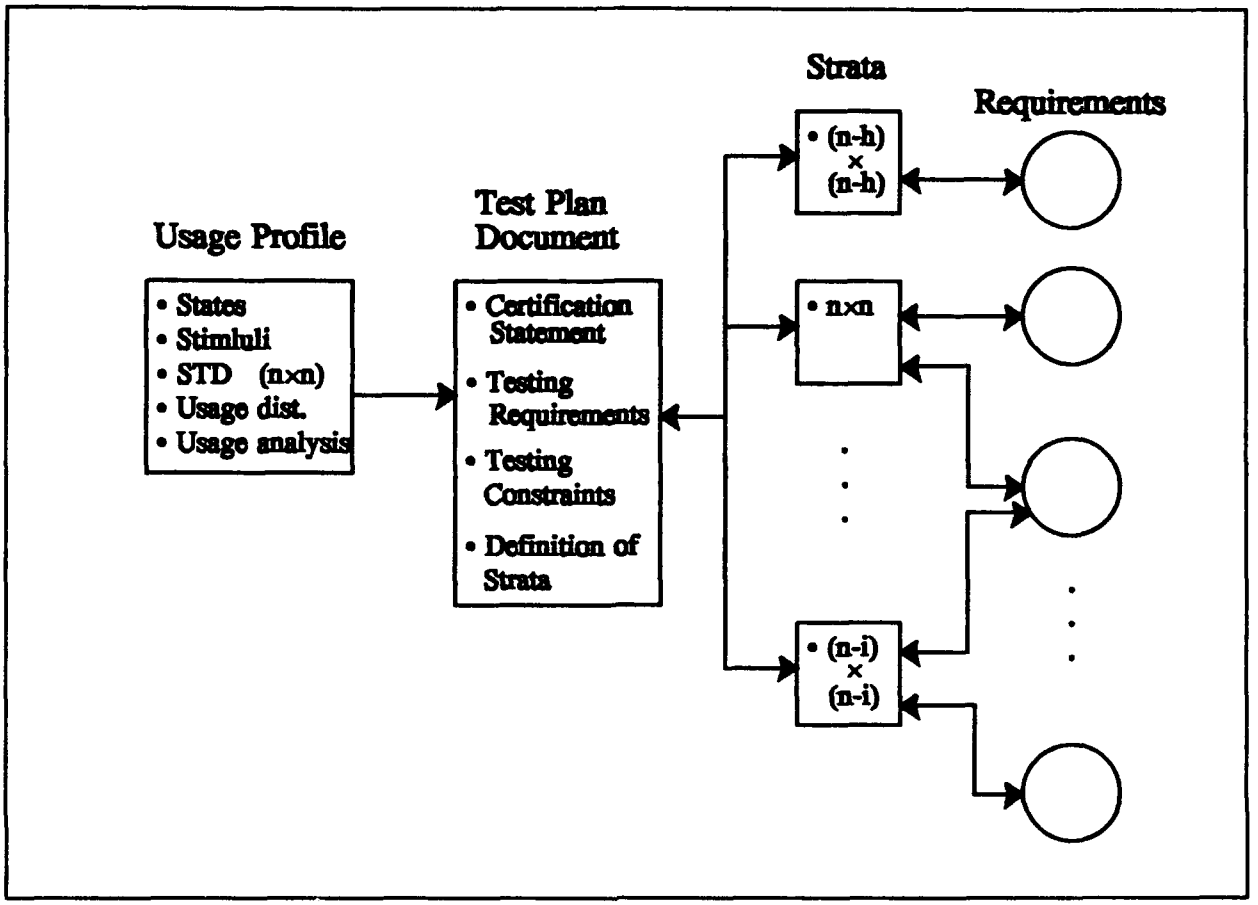
A schematic of the test planning phase is picture in Figure 3.2.1. Note that this diagram defines an accumulation of size $n \times n$ and strata of smaller sizes ($n\text{-}h \times n\text{-}h$ and $n\text{-}i \times n\text{-}i$) as well as a stratum of the same size as the accumulation model. All of these definitions are legal under the certification framework. The stratum of the same size as the entire accumulation profile indicates that the model has the same states and arcs but defines different probabilities. This is a common occurrence in software certification for different user classes with the same possibility space but different usage characteristics.

3.3 Prepare Test Scenarios or Test Scenario Generator for Accumulation j (C5.j.2.2)

The number of test scenarios required to certify an increment is directly related to the complexity of the usage profile defined as a Markov process. From the Markov chain, certifiers compute the expected number of test scenarios until each stimulus is generated at least once. This number is excellent insight into the *expected* minimum amount of testing required to test the increment.

Test generation from the usage chain is achieved by traversing the arcs of the usage chain from invocation to termination (according to the definition of a use) based on the transition probabilities. This is easily automated using a good random number generator and any high level language. The Certification Assistant tool automatically generates these stimuli sequences. As each arc is traversed, any necessary information is included in a test scenario file. The information included in the scenario file is determined via *test fragments* which correspond to the arcs of the chain. Most often, the test fragments simply output a single stimulus label or a series of labels. Thus the test scenario is a sequence of stimuli that the certifier applies to the software,

Figure 3.2.1: The Test Planning Process



in the order given, when the accumulation is available for certification. Some items that are useful information to included in the test fragments are listed below.

- state labels (operational software state)
- arc labels (stimuli)
- responses, pre-conditions, post conditions
- requirements met
- any other information that will be helpful in executing the scenario

The analytical results help certifiers discover capabilities and properties of the usage model which are not directly apparent from the model itself and to make predictions about how the test will unfold, e.g., the expected amount of testing until all the stimuli appear at least once, etc. If the values computed analytically do not match some known usage probabilities or intuition, then an investigation into (and possibly re-estimation of) the appropriate transition probabilities is merited. Furthermore, since certifiers are able to statistically assess the amount of testing necessary to get appropriate domain coverage and usage representation, they are able to formulate a testing budget based on information obtained through scientific analysis.

A sample of two test scenarios from the example usage profile in section 2.2 of this volume appear in Table 3.3.1. In this example, the test scenario consists of a sequence of external stimuli, no states are necessary to execute the scenarios.

Table 3.3.1: Test Scenarios and Expected Results

Sequence Number 1:	Expected Response 1:	Sequence Number 2:	Expected Response 2:
CRM <legal_filename>	name in project field	CRM	project field blank
Dn Arrow key	cursor on line 2	Dn Arrow key	cursor on line 2
Enter key	display screen 3, back to screen 1	Enter key	display screen 3, back to screen 1
Enter key	display screen 3, back to screen 1	Dn Arrow key	cursor on line 3
Enter key	display screen 3, back to screen 1	Up Arrow key	cursor on line 2
Dn Arrow key	cursor on line 3	Up Arrow key	cursor on line 1
Up Arrow key	cursor on line 2	Up Arrow key	cursor on line 5
Dn Arrow key	cursor on line 3	Dn Arrow key	cursor on line 1
Enter key	display screen 4, back to screen 1	Enter key	display screen 2, back to screen 1
Dn Arrow key	cursor on line 4	Enter key	display screen 2, back to screen 1
Up Arrow key	cursor on line 3	Up Arrow key	cursor on line 5
Enter key	display screen 4, back to screen 1	Enter key	exit to OS
Dn Arrow key	cursor on line 4		
Enter key	display screen 5, back to screen 1		
Dn Arrow key	cursor on line 5		
Up Arrow key	cursor on line 4		
Dn Arrow key	cursor on line 5		
Enter key	exit to OS		

3.4 Determine Expected Test Results (C5.j.2.3)

Determining the expected results for the test sequences is a difficult manual step for which we can, unfortunately, offer no profound wisdom as guidance. Basically, one must use the specification as the oracle for determining the expected external responses and behavior of each test scenario. It is a good practice to record these responses (in the form of screen display, changes to data files, etc.) in advance to ease the burden of comparing the actual and expected result for each test scenario.

An example of expected results is included in Table 3.3.1.

3.5 Increase Understanding of Problem and Solution Domains (C5.j.2.4)

Many issues and questions arise during all phases of the Cleanroom certification process. These issues are often the result of some new, unknown factor being noticed. This factor may become a part of the certification activity or it may not. But to make any sort of determination, the issue must be understood. As a result, the Certification Team must find clarifications and answers expeditiously.

The first task required of the Certification Team when faced with a issue or question is to fully understand it. This will require talking with the individual(s) who raised the problem. If it is a simple misunderstanding, the matter can be settled by a clarification and complete answer.

Deeper issues will necessitate an increased understanding of the problem domain and the solution domain. A Certification Team member, or the entire Certification Team may be required to go off and perform an analysis task. This will typically include information through interviews and document reviews. Also, more information on solution opportunities can be gathered. This new information is integrated with the known information to create an analysis report that documents what has been learned. The task of increasing the understanding of system requirements continues until the resolutions to the questions are considered satisfactory.

SECTION 4: SOFTWARE CERTIFICATION FOR ACCUMULATION J TASKS

Process P6.j, Software Certification for Accumulation j consists of six tasks with the purpose of gathering and organizing the data that is necessary to perform a scientific certification of the quality of a product accumulation. These tasks result in the certification of the software developed to the specification, allowing the project team to appraise the quality of the software product and of the software development process.

```
proc P6.j: Software Certification for Increment/Accumulation j
   [P6.j certifies the code, and makes the decision as to whether an increment will be accepted
   or rejected.]
   do   [P6.j: Software Certification for Increment/Accumulation j]
      C6.j.1: Build Accumulation j;
      if no pre-certification failures
      then
         while
            certification plan requires more tests and sufficient failures have not been observed
            to make it desirable to terminate testing and wait for corrections
         do
            C6.j.2: Perform Certification Tests for Accumulation j;
         od;
      fi;
      if at least one failure observed and observed failures are considered to be correctable
      then
         C6.j.3: Prepare failure report(s);
         D6.j.4: Correct failure, verify correction and prepare ECN;
      fi;
      M6.j.5: Management Decision: (1) certification complete-accumulation quality satisfactory,
              (2) certification complete-quality not satisfactory-replanning Is required or (3)
              certification should continue;
      if not Management Decision is certification should continue
      then
         C6.j.6: Prepare Certification Report(s);
      fi;
   until
      Completion Conditions achieved for C6.j.6
   od;
corp;
```

Of the six subtasks, two of them, D6.j.3, Correct failure, verify correction and prepare ECN, and M6.j.6, Management Decision: (1) Certification Complete-Accumulation Quality Satisfactory, (2) Certification Complete-Quality Not Satisfactory-Replanning Is Required or (3) Certification

Should Continue, are described in detail in separate volumes. D6.j.3 is explained in Volume 5, section 5 and M6.j.6 is described in Volume 3. The other tasks are described in detail in the sections below.

4.1 Build Accumulation j (C6.j.1)

Before certification can begin, the Certification Team must received a fully verified code accumulation from the Development Team.

When an accumulation is available from the Development Team, it is given to the Certification Team for compilation, linkage, and execution. The Certification Team owns the accumulation and the compiler at this point and the software is under strict configuration control. The software being under configuration control means that all failures and engineering changes are formally documented. The Certification Team places the software into the controlled library and compiles each code unit in the accumulation. Upon completion of successful compilation, the code is linked or assembled. Once the code is ready for execution, it is placed in the execution library so that test scenarios may be run. The process for compilation and linking of Cleanroom code is not substantially different than for other software development processes. The primary differences are the individuals who conduct the compilation and linking (i.e., the Certification Team as opposed to Development Team) and the early appearance of code under configuration control (code is placed in a controlled library before first compilation as opposed to before system integration).

4.2 Perform Certification Tests For Accumulation j (C6.j.2)

When the usage chains for the strata are complete and the analytical results are acceptable, test scenarios are generated and certification begins. The certification for each stratum can be performed in parallel. However, each such test must be performed on the same version of the code. Thus when failures reports are submitted to the Development Team from one certifier, all test activity should be halted until the engineering changes are complete and the new software version is available.

It is crucial to enforce that only the statistical test scenarios be executed, i.e., no other executions, for purposes of finding failures, against the software are allowed until the desired level of certification has been reached. At this time some crafted test cases can be executed, but the results are not included in the certification report. However, any failures found during such testing must be reported (along with all the other failures found during execution) in a final Accumulation Failure Report. This report will contain all compilation failures, link failures, and execution failures that were observed.

Test scenarios are executed either manually or automatically depending on the environment and the automated support available. Each stimulus which appears in the scenario must be executed, including those that represent illegal behavior. Tests are executed in the order that they are generated from the model and each is checked against the specification which acts as the oracle

for the certification. It is a good practice to analyze the test scenarios in advance to determine the expected behavior and use this to aid in the evaluation effort when executing the tests.

Upon the observation of a failure, a decision must be made to either halt testing while the code is re-engineered or to continue testing and postpone the repair. Catastrophic failures require an immediate halt because they effectively inhibit the ability to execute further tests. Non-catastrophic failures may be queued for repair as long as they do not affect the ability to execute the test and construct the certification history. Local work rules must be established to determine the action to be taken in both circumstances. Typically, returning non-catastrophic failures to the developers as they occur (or in groups) is acceptable. Certification can continue while these failures are repaired, however, catastrophic failures require that all certification activity halt while corrections are made.

Failures can be classified according to two general criteria: reasonable mistakes and basic design flaws. Software development is a creative process that can lead to the introduction of errors. Even though Cleanroom is a rigorous engineering process, humans make mistakes and these mistakes are understandable. However, Cleanroom also offers software engineers the ability to create correct and efficient designs. Thus, errors in design, that should not have persisted through a box structure expansion and verification, are not easily overlooked. When several of these types of errors are located in a statistical test, redesign is more effective at improving the software than additional testing and the software should be returned to the developers for rework.

Failures are reported to the Development Team via a failure report which gives sufficient information for the developers to use in code walkthroughs to locate and fix the error.

Upon verification of the changes, an engineering change notice is submitted to the Certification Team so that the code changes can be instituted and recompilation can occur.

A testing log must be created for each stratum in the accumulation. A sample testing log appears in Table 4.2.1.

Table 4.2.1: A Sample Testing Log

Project:_____

Date	Time	Accum. #	Version #	Certifier Name	Scenario # (or C/L)	Failure Report #	States Traversed	Cont. after failure?

4.3 Prepare Failure Reports

In the event that compilation or link failures occur, formal documentation is created to record the appropriate information. A compilation or link failure is any action that results in the compiler or link editor terminating unsuccessfully. A failure report must be completed for each failure observed, i.e., for each problem the compiler or link editor finds. A sample of the failure report form appears in Table 4.3.1 below. The failure report should provide sufficient information to the Development Team to allow them to isolate and correct the failure. All failure reports from the compilation or link editing of a subroutine or a set of subroutines should be filled out and returned to the Development Team as a set. All compilation or link failures must be corrected (and the changes verified) before certification can proceed.

Table 4.3.1: An Example Failure Report

FAILURE REPORT

Number:___ Project:_____ Increment:___
Certifier:_____ Date:_____

Failure During: Compilation: ___
Linking: ___
Execution: ___ [Failure #_____ Test Case #_____
Transition #_____ Version #_____]

Briefly describe the failure:

_____
_____
_____
_____

Attach any documentation that may clarify the failure:

_____

Approval: _____ _____ _____

Any failures observed during compilation or linking should be categorized as residual design defects and counted in the final errors/KLOC. In all cases, a failure report should be included with the certification statistics. A general rule of thumb in Cleanroom is that greater than 5 errors/KLOC is not sufficient quality and the code should be redesigned and reverified instead of performing additional testing. However, each situation should receive individual consideration and many merit a different strategy. For example, new Cleanroom teams and teams working with new implementation languages often need a period of methodology and language familiarization before such a rule can be strictly enforced. But in any situation, a large number of serious errors that result from poor design and/or illogical thinking bring into question the process that created the software and redesign is necessary.

The compilation failures are investigated by the Development Team based on the failure report and code inspection. Any changes must be team verified. An engineering change notice is then submitted to the Certification Team for the actual changes to the code to be made. The Certification Team retains control of the code throughout this process. Once the changes are made, the Development Team is consulted for permission to recompile.

Failure reports are also completed for execution failures as well. Failure reports are typically grouped and returned to the Development Team for correction as a set. Usually these reports are accumulated either until (1) a specific test suite has been executed, (2) a specified number of failures have been found, or (3) a failure has been observed that is serious enough, in terms of its effect on functionality, to jeopardize further testing. Of course the certification report (describes in the next section) can be used to indicate the return on investment of additional testing. If no substantial rise in the reliability is indicated, then it is wise to halt testing and submit failure reports for engineering changes. When failure reports have been submitted to the Development Team for correction, certification can proceed in some cases. Other circumstances merit a halt to testing activity until all of the failures are resolved, either by correction or otherwise. The failures, as well as successes, are noted to calculate the level of certification for the accumulation.

4.4 Prepare Certification Report(s) (C6.j.3)

The Certification Report is a document that contains the data from the entire software testing process. This includes any statistical reports and testing logs as well as any other important team experiences that characterize certification of a particular accumulation. Often, important insights are gained that enable future teams to perform better tests. Such data should be documented so it will be useful to any appropriate stakeholder.

One of the most important ingredients of the certification report is the team's recommendation for acceptance/release of the product.

The Certification Team can choose from any number of data reporting strategies that exist in the literature to include in the certification report. Cleanroom can support a wide range of reliability modeling and/or growth modeling. Most models require only the number of test cases that were

applied to the software, along with the number that failed, to be input to the model. This data is then used, often accompanied by assumptions about its probabilistic makeup, to estimate the system reliability and *MTTF*. For example, hypothesis testing assumes that failures are discovered according to a binomial distribution. Additionally, the certification model (a popular reliability-growth model) computes the reliability as a function of the interfail times of the software assuming that the interfail times are exponentially distributed. It is wise to use a variety of methods so that a basis of comparison across models and across projects is established. In the following sections, several options are outlined.

4.4.1 Markov Chain Reliability Model

A method for modeling software reliability and/or correctness using all the data from the statistical test, including each individual stimulus applied and each operational state visited has been developed. This model allows a detailed certification that is sensitive to the usage distribution (as defined by the usage profile) and does not make assumptions about the distribution of important random variables.

As the test progresses, the *certification history* is maintained as a sequence of stimuli that were actually executed against the software, i.e., the certification history is the ensemble set of test scenarios actually executed and treated as a single long sequence. When failures are observed, a unique symbol that marks each the failure is inserted into the certification sequence immediately following the last stimulus successfully applied to the software. Note that if a failure causes parts of the scenario to be impossible to execute (as happens with a system crash) then these states are not included in the certification history. Thus, the certification history is a sequence of actual stimuli applied to the software with failure symbols included at the location they occurred.

Let s_1, s_2, ..., s_n denote the sequences of stimuli generated from the usage model and applied to the software. If no failures are discovered, the certification history (i.e., the unmodified sequences) is encapsulated into a model called the *testing model* (or, equivalently, the *testing Markov chain*). The initial testing model T_0 is an exact copy of usage chain with all arc probabilities set to zero. Testing chain T_1 is obtained from T_0 by incrementing arc frequencies along the path of states in s_1. Similarly, T_2 is obtained from T_1 by incrementing the arc frequencies traversed during sequence s_2. In general, T_i is obtained from T_{i-1} by incrementing the arc frequencies which exist in sequence s_i.

Frequency counts in testing chain T_i are always obtained from specific sequences applied to the software; making the testing chain an accurate picture of the current state of the certification process. At the completion of any sequence i during the certification process, the frequency counts can be converted to probabilities whenever computation using T is required. The testing chain can be used to determine the degree to which the test cases generated are representative of the original usage model. This is performed through a statistical comparison of the usage model and the testing model.

If the certification history contains failures, states are added to T to model the failures. Consider the j^{th} failure $\mathbf{f}_j$ detected during input of sequence s_i. A new state labeled $\mathbf{f}_j$ is placed in T_i at the exact location of its occurrence in s_i. The arcs to and from $\mathbf{f}_j$ are assigned frequency count 1. In the event that j is catastrophic, then the arc from $\mathbf{f}_j$ goes to the terminate state; otherwise the application of the test sequence can continue and the arc from $\mathbf{f}_j$ goes to the next state in s_i. The testing chain T_i incorporates the underlying structure of the usage profile and the failure characteristics of the test history; making it a powerful and accurate computational device; as will be described in the next section.

A testing model is created for each stratum defined for the accumulation.

The Certification Assistant tool automatically constructs certification histories.

A testing model is the basis for an analysis which yields very valuable information.

First, analytical stopping criteria are available through comparison of the statistical properties of the usage profile (what is expected to occur in test) and the testing model (what did occur in test). When these properties are sufficiently similar then testing can stop because the test sequences used are a representative sample of the usage model itself. These measures are called discriminants.

Two discriminant functions, the *discrimination* and *distance*, are available for incorporation into any certification experiment.

The discrimination measure describes the difference between U and T in terms of the their stochastic sequence generating properties. When the discrimination is 0, U and T generate the same set of sequences. When the discriminant is small, the sequences generated are very similar. When T is large or infinite, the sequences are very different indeed. A small discrimination means that U and T generate a similar sequence set. In order to do this, parameters of U and T must be very similar, indicating that T is representative of U. The discrimination is only computable when all of the arcs in U have been traversed at least once during testing.

The distance is a measure related to the discrimination in that it is based on a comparison of U and T. The distance is computed by a geometric interpretation of U and T in multi-dimensional euclidean space. U represents a fixed vector in space and T changes as it is updated during the test. The euclidean distance is defined for any T_i and U indicating how distant the vectors are and indicating to the certification team the representativeness of the sample.

We recommend that both measures be computes with each sequence applied to the software so that certifiers have multiple datum to use in the determination of stopping the certification process.

The Certification Assistant tool computes both of these measures.

Second, software quality measurements are available from the testing model. Computable values include the following.

- reliability: the probability of correct execution
- *MTTF*: the expected time that the software will run from invocation
- *MTBF*: the expected time that the software will run between failures
- failure rate: the expected number of failures per execution

The Certification Assistant tool computes each of these measures from the testing model.

The construction of the testing model is an algorithm. However, several options exist for determining what data to store in the testing model and when to re-initialize it. Two major decisions must be made. First, should errors be counted every time they are encountered or should they be acknowledged only at their first occurrence. In other words, do we include information about repeated software failure which corresponds to the failure reports or simply acknowledge each individual failure corresponding to an engineering change notice. Second, after the software has be re-engineered by the Development Team as a result of the failures, should the testing chain be re-initialized at zero frequency counts in order to mark the certification of a new software version or should the history of the accumulation be maintained.

There are two different strategies for reporting test results. The first strategy keeps all the testing data for an entire accumulation in a single testing model and counts failures only once. This *accumulation testing model* describes the entire testing experience of an accumulation and allows analysis of the performance of the software over the entire accumulation. In this model, each failure state is installed only once and the frequency counts into the failure states are not updated unless they appear again after an attempt was made to repair them. In other words, a state is installed for each engineering change applied to the software.

The second strategy is to begin construction of the testing model, adding failure states as appropriate, and restart the frequency counts at zero after modifications to the software are made and testing resumes. Note that the failure states are removed when the frequencies are re-initialized. In this model, frequency counts to the failure states are updated each time the failure state is observed. Thus, a separate testing model for each version within an accumulation is established. This strategy is called the *version testing model* and consists of a single model for each version within an accumulation. Of course, other models are possible and may be valuable for specific applications.

Computation of the analytical results from the testing model is the same regardless of which model (accumulation model or version model) one decides to build. However, the interpretation of the results are different. Analysis of the accumulation testing model yields data about the process of testing the code. The results apply to the overall process and the individual failures within the process. Failure states are not physically removed but their effect is washed out by repeated successful sequences, giving evidence that the failures were successfully eliminated by the engineering changes. The version testing model is an accurate picture of the current state of

the software and the results apply only to a specific version. Table 4.4.1 summarizes the organization and interpretation of each of the two methods of software certification.

Table 4.4.1: Two Views of Software Certification

Strategy	Rules	Analysis
Accumulation Testing Model	Each test sequence from the entire accumulation is included Frequency counts are never re-initialized Failure states are never removed Frequency counts to failure states are incremented only on the first occurrence of the failure *or* after an attempt has been made to correct the failure and it re-appears	Results apply to the *testing process* and not necessarily to the software The *reliability* (and other quality measures) reflects the entire testing process and every failure observed in the accumulation The *discriminant* functions reflect the restoration of confidence in the software despite the appearance of failures
Version Testing Model	Only the test sequences for a specific version are included Frequency counts are re-initialized when changes to the software are made (i.e., a new version is obtained) Failure states are removed when changes to the software are made Frequency counts to failure states are updated each time the failure is observed	Results are applicable to the *current version* of the software if it were released to users without change The *reliability* reflects the current status of the version of code currently in testing The *discriminant* functions reflect the progress of testing on the current version

A sample certification report appears in Table 4.4.2 for an accumulation. The certification reports for the individual versions appear in Tables 4.4.3 to 4.4.5. These tables illustrate the differences between two interpretations of the test data. This data shows a total of four failures observed during the application of 50 test sequences. Table 4.4.2 contains the computations from the accumulation interpretation of the testing model. Note that since failures have occurred, this interpretation will never yield an estimate that the software is correct despite changes to the code that may cause it to be so. Also, each failure appears only once, signifying that the failures were not encountered after the engineering changes were implemented. The subsequent tables shows

the computations according to each version. The first version (Table 4.4.3) of the software contains three failures in 15 sequences. Multiple appearances of the failures are also logged. After sequence 15, failure reports are submitted and the software is re-engineering. Testing is restarted at this point on the second version of the software. The fourth failure is observed at this point, occurring three time before testing is again halted. The last version then reveals failure-free behavior over 25 additional sequence resulting in a reliability estimate of 1. Thus, the stopping criteria are the measurement that become the indicator of how much evidence the statistical test has given that support this conclusion.

These report(s) serve two distinct purposes. First, in their complete form shown in Tables 4.4.3 to 4.4.5, they give detailed information to the developers and certifiers who are working on the project. Additionally, graphing the data is also helpful in discovering general trends and compactly displaying the results. Second, summary reports are good for other stakeholders on the project. Summary reports contain at least the following items.

1. the total number of failures in each version of each stratum
2. the total number of engineering changes in each version of each stratum
3. last line of each certification report containing the reliability, etc.

The Certification Assistant tool produces reports similar to the following.

Table 4.4.2: Certification Statistics for a Sample Accumulation

Sequence Number	Reliability	*MTBF*	Discrimination	Distance
1	1.0	∞	∞	0.90459634
2	1.0	∞	∞	0.90663380
3 F1	0.7621622	163.0	∞	0.88497270
4	0.8216216	174.0	∞	0.74395470
5	0.8572973	214.0	∞	0.60730044
6	0.8736360	220.0	∞	0.61295286
7 F2	0.7488309	124.4	∞	0.67786315
8	0.7802270	130.0	∞	0.49215428
9	0.7992595	175.0	∞	0.49382823
10	0.8171245	178.0	∞	0.49754267

11 F3	0.7649534	125.3	∞	0.57682800
12	0.7800361	136.3	∞	0.57173108
13	0.7954131	142.0	∞	0.57280911
14	0.8100399	158.0	∞	0.57627548
15	0.8216386	169.0	∞	0.57726050
16 F4	0.7815236	128.0	∞	0.65294606
17	0.7928913	133.5	∞	0.64848599
18	0.8018171	139.3	0.01831863	0.64201852
19	0.8105890	140.2	0.01772956	0.64070279
20	0.8195415	141.8	0.01797155	0.64153931
21	0.8266051	144.0	0.01770270	0.64037591
22	0.8341026	145.5	0.01798910	0.64126912
23	0.8404136	148.3	0.01332975	0.63638060
24	0.8466884	150.5	0.01359349	0.63742170
25	0.8520939	152.0	0.01354661	0.63699330
26	0.8575536	153.5	0.01386466	0.63793228
27	0.8620913	156.2	0.01395083	0.63740685
28	0.8675527	165.8	0.01088227	0.63434016
29	0.8715451	170.5	0.00927842	0.63150189
30	0.8755806	180.3	0.00971495	0.63287873
31	0.8793332	187.5	0.00927639	0.63197463
32	0.8827737	188.5	0.00919315	0.63173014
33	0.8861242	196.3	0.00929397	0.63167292
34	0.8891583	200.0	0.00793337	0.62892664
35	0.8921083	201.0	0.00791309	0.62886426
36	0.8947823	232.7	0.00773257	0.62937304
37	0.8975135	238.8	0.00753182	0.62936450
38	0.9002818	247.0	0.00704949	0.62853974
39	0.9026693	248.0	0.00700537	0.62841527
40	0.9050651	255.5	0.00682846	0.62857363

41	0.9072665	262.2	0.00706612	0.62913210
42	0.9093364	263.2	0.00703667	0.62905020
43	0.9114076	270.0	0.00705158	0.62903299
44	0.9133771	291.0	0.00633379	0.62628399
45	0.9151931	295.7	0.00626138	0.62588795
46	0.9169896	307.2	0.00661746	0.62584311
47	0.9187180	319.0	0.00698431	0.62621041
48	0.9203730	333.5	0.00735684	0.62661352
49	0.9219189	334.5	0.00726743	0.62641310
50	0.9233891	342.5	0.00715505	0.62649094

Summary:

Total Scenarios Executed: 50

Total Number of Failures: 4

Total Number of Engineering Changes: 4

Graphical Representations may also be included to put the overall picture in perspective. Figures 4.4.1.1 to 4.4.1.4 represent each of the above measures graphically. The graphs help make explicit data trends that are hard to see from long lists of numbers.

Figure 4.4.4.1: Graphical Representation of the Reliability

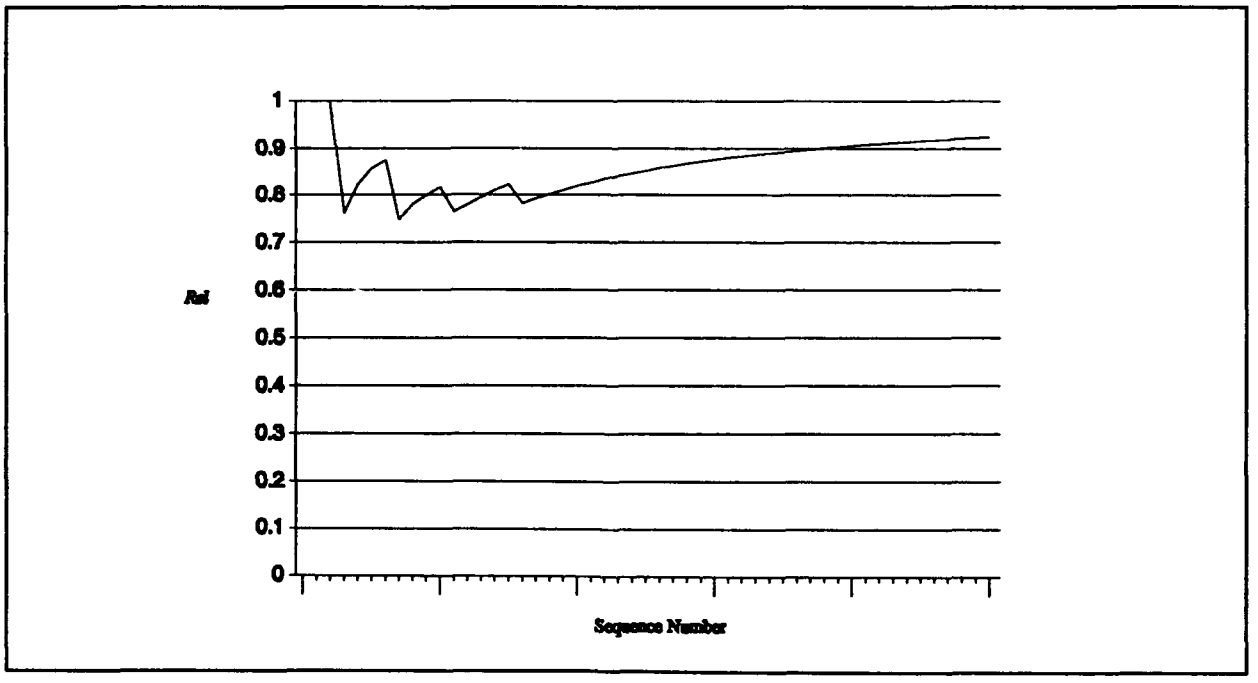


Figure 4.4.4.2: Graphical Representation of the *MTBF*

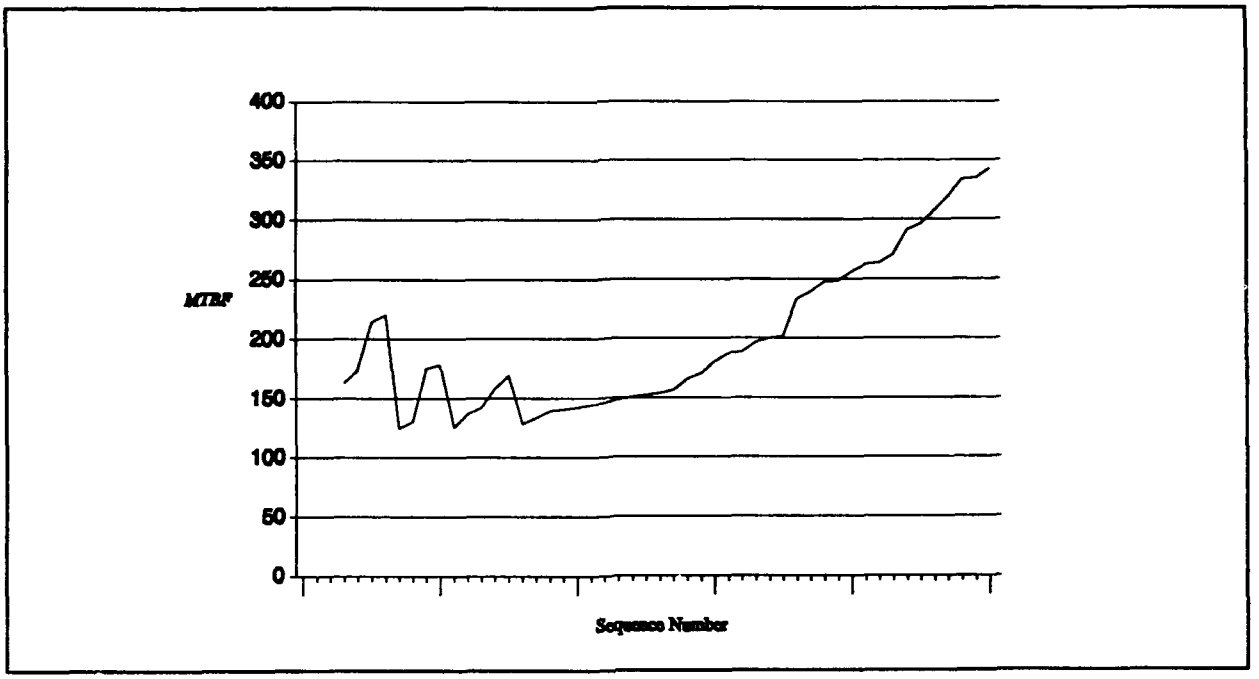


Figure 4.4.4.3: Graphical Representation of the Discrimination

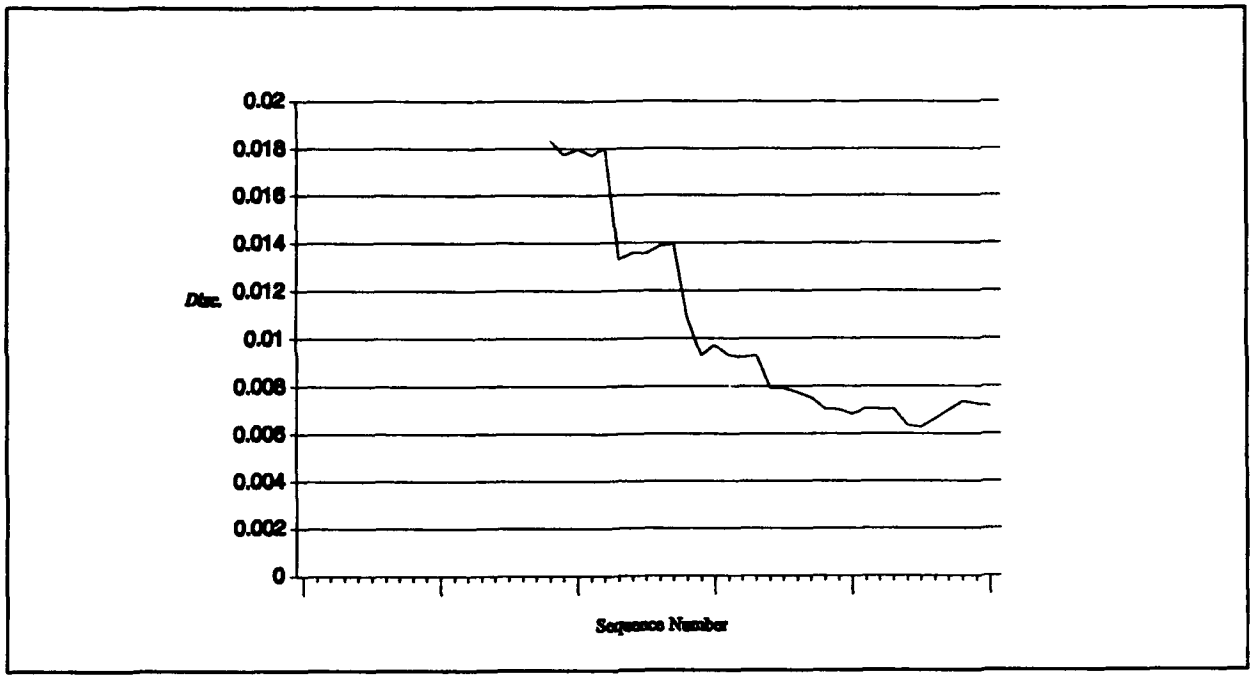


Figure 4.4.4.4: Graphical Representation of the Distance

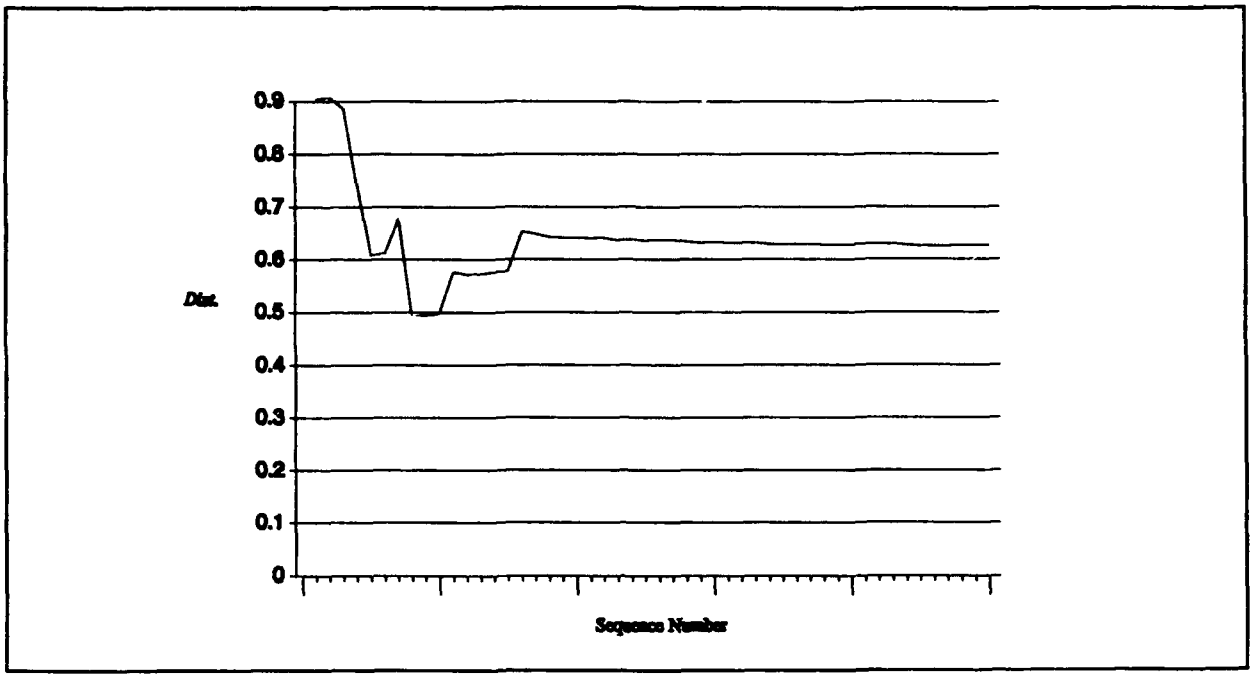


Table 4.4.3: Certification Statistics for Version 1

Sequence Number	Reliability	*MTBF*	Discrimination	Distance
1	1.0	∞	∞	0.90459634
2	1.0	∞	∞	0.90663380
3 F1	0.7621622	163.0	∞	0.88497270
4	0.8216216	174.0	∞	0.74395470
5	0.8572973	214.0	∞	0.60730044
6	0.8736360	220.0	∞	0.61295286
7 F2	0.7488309	124.4	∞	0.67786315
8 F1	0.7052270	87.0	∞	0.57896791
9	0.7325929	117.0	∞	0.58039152
10 F1	0.6873774	89.5	∞	0.66092179

11 F3	0.6546406	75.6	∞	0.72362404
12	0.6737854	82.2	∞	0.71989707
13	0.6944773	85.6	∞	0.72074781
14	0.7124546	95.2	∞	0.72374514
15 F2	0.6730047	80.7	∞	0.78911525

Summary:

Total Scenarios Executed: 15

Total Number of Failures: 3

Total Number of Engineering Changes: 3

Table 4.4.4: Certification Statistics for Version 2

Sequence Number	Reliability	*MTBF*	Discrimination	Distance
1 F4	0.5000000	15.0	∞	0.98381352
2	0.7500000	19.0	∞	0.95566751
3	0.8333333	128.8	∞	0.67827290
4	0.8333333	103.0	∞	0.56749205
5	0.8666667	122.0	∞	0.42509080
6 F4	0.7627628	66.0	∞	0.53881485
7	0.7847490	79.5	∞	0.51760079
8	0.8054054	84.5	∞	0.51899412
9	0.8265700	86.5	∞	0.51132867
10 F4	0.7805166	64.0	∞	0.59302440

Summary:

Total Scenarios Executed: 10

Total Number of Failures: 1

Total Number of Engineering Changes: 1

Table 4.4.5: Certification Statistics for Version 3

Sequence Number	Reliability	*MTBF*	Discrimination	Distance

1	1.0	∞	∞	0.93185033
2	1.0	∞	∞	0.90208448
3	1.0	∞	∞	0.60042497
4	1.0	∞	∞	0.46935659
5	1.0	∞	∞	0.28098143
6	1.0	∞	∞	0.26613835
7	1.0	∞	∞	0.22884672
8	1.0	∞	∞	0.23306323
9	1.0	∞	∞	0.21545436
10	1.0	∞	∞	0.19830691
11	1.0	∞	∞	0.17772825
12	1.0	∞	∞	0.19061005
13	1.0	∞	∞	0.18447346
14	1.0	∞	∞	0.18494891
15	1.0	∞	∞	0.18974892
16	1.0	∞	∞	0.17355669
17	1.0	∞	∞	0.17578204
18	1.0	∞	∞	0.18049896
19	1.0	∞	∞	0.18322122
20	1.0	∞	∞	0.18550743
21	1.0	∞	∞	0.17021599
22	1.0	∞	∞	0.16099319
23	1.0	∞	0.01005985	0.14324902
24	1.0	∞	0.00576479	0.12043814
25	1.0	∞	0.00600560	0.12399226

Summary:

Total Scenarios Executed: 25
Total Number of Failures: 0
Total Number of Engineering Changes: 0

The interpretation of this data is an important factor in the success of Cleanroom certification. The data from the entire certification history is important for analyzing the *testing process* itself. That is how has the entire experience of uncovering and fixing errors affected the quality of the code and the representativeness of the sample? The individual version chains help to quantify the behavior of the actual software product. In these chains, each failure is noted each time it occurs. Thus a reliability curve that is very flat will indicate that engineering changes are necessary in order to certify.

Each project is different. They each have their own challenges and analyses. Experimentation with the methods through the Certification Assistant is an excellent way to gain important analytical experience with the Markov chain reliability model.

4.4.2 Hypothesis Testing

The purpose of hypothesis testing of software *P* is to determine the number of test cases (sample size to make the statement that *P* is at least *r%* reliable with *c%* confidence. Table 4.4.2.1 summarizes some example values for *r%* reliability and *c%* confidence.

Table 4.4.2.1 Sample Sizes for Hypothesis Testing

	Confidence Level			
Reliability	90%	95%	99%	99.9%
.90	22	29	44	66
.95	45	59	90	135
.99	230	299	459	688
.999	2302	2995	4603	6905
.9999	23025	29956	46049	69074

4.4.3 The Certification Model (Reliability-Growth Modeling)

The certification model is based on the assumption that the interfail times for software are exponentially distributed as engineering changes are applied to software. This assumes that no new failures are introduced when the fault is corrected and that the exponential distribution is an accurate description of the failure characteristics of the software.

This second assumption is empirically supported by some research done by Adams on nine large IBM software projects. Mills argued that this data depicts an exponential growth in the *MTTF* as bugs are removed from the software.

In order to use the certification model. One collects the interfail times of the software during a statistical test and plots these items on a graph with the time (which is normally measured using test case number) as the *x* coordinate and the interfail value as the *y* coordinate. A corrected log least squares curve fit is then applied to the series of points to predict the location of the next data point.

4.4.4 Bayesian Estimation

Bayesian estimation is a controversial idea which is not generally accepted in the engineering community. This theory allows the use of "outside knowledge" about the probability of software failure to be included in the certification results. The method consists of choosing values for parameters *a* and *b* based on knowledge about the probability of failure and then executing *n* successful test cases. The parameter *a* contributes negatively (the knowledge that makes the software worse) and parameter *b* contributes positively (the knowledge that makes

the software better). The formula: $a/n+a+b$ is then used to calculate the probability of failure for the software.